International Journal of Research in Computer Science
eISSN 2249-8265 Volume 2 Issue 6 (2012) pp. 33-37
www.ijorcs.org, A Unit of White Globe Publications
doi: 10.7815/ijorcs. 26.2012.053

COMMENTZ-WALTER: ANY BETTER THAN AHO-CORASICK FOR PEPTIDE IDENTIFICATION?

S.M. Vidanagamachchi[1], S.D. Dewasurendra[2], R.G. Ragel[2], M.Niranjan[3]

[1]*Department of Computer Science, Faculty of Science, University of Ruhuna, Matara, SRI LANKA*
Email: smv@dcs.ruh.ac.lk

[2]*Department of Computer Engineering, Faculty of Engineering, University of Peradeniya, SRI LANKA*
Email: devapriyad@pdn.ac.lk, roshanr@pdn.ac.lk

[3]*Department of Electronics and Computer Science, Faculty of Physical and Applied Sciences, University of Southampton, UNITED KINGDOM*
Email: mn@ecs.soton.ac.uk

Abstract: *An algorithm for locating all occurrences of a finite number of keywords in an arbitrary string, also known as multiple strings matching, is commonly required in information retrieval (such as sequence analysis, evolutionary biological studies, gene/protein identification and network intrusion detection) and text editing applications. Although Aho-Corasick was one of the commonly used exact multiple strings matching algorithm, Commentz-Walter has been introduced as a better alternative in the recent past. Comments-Walter algorithm combines ideas from both Aho-Corasick and Boyer Moore. Large scale rapid and accurate peptide identification is critical in computational proteomics. In this paper, we have critically analyzed the time complexity of Aho-Corasick and Commentz-Walter for their suitability in large scale peptide identification. According to the results we obtained for our dataset, we conclude that Aho-Corasick is performing better than Commentz-Walter as opposed to the common beliefs.*

Keywords: *Aho-Corasick, Commentz-Walter, Peptide Identification*

I. INTRODUCTION

String matching has been extensively studied in the past three decades and is a classical problem fundamental to many applications that need processing of either text data or some sequence data. Many string matching algorithms are used in locating one or several string patterns that are found within a larger text. For a given text T and a pattern set P, string matching can be defined as finding all occurrences of pattern P in text T.

With the remarkable increase of data in biological databases (such as what is available in GenBank and UniProt) there exists a bottleneck in analyzing biological sequences. Pair wise sequence alignment, multiple sequence alignment, global alignment and local alignment are types of string matching techniques used in computational biology for matching biological sequences [1]. Multiple Sequence Alignment (MSA) is an alignment of three or more sequences that contain thousands of residues (nucleotides or amino acids). Multiple pattern matching plays a vital role in identifying thousands of patterns simultaneously within a given text. The results of MSA can be used to infer sequence homology as well as conduct phylogenetic analysis and multiple sequence matching is used for identifying proteins/ DNA/ Peptides/ motifs/ domains, managing security (for example, intrusion detection), editing texts, etc. In multiple sequence alignment we use a database of large sizes of sequence data over a query sequence and it can be identified as one form of multiple string/pattern matching. In this paper, we have used a set of predefined peptides and located them over a given arbitrary protein (the protein can be known/ unknown/ predicted).

Exact (as opposed to approximate) string matching is commonly required in many fields, as it provides practical solutions to the real world problems. For example, exact string matching can be used for sequence analysis, gene finding, evolutionary biology studies and protein expression analysis in the fields of genetics and computational biology [8][15][16]. Other areas are network intrusion detection [17][18][19] and text processing [20].

In this paper we basically compare time complexities for two multiple pattern matching algorithms: Aho- Corasick and Commentz-Walter for the identification of large scale protein data.

We start by giving brief introduction of string matching algorithms in section II. Section III includes the objectives of the experiment. Related work, experimental set up and methods, results of the experiments, conclusion and future scope are discussed in section IV, section V, section VI, section VII and sections VIII respectively.

II. STRING MATCHING ALGORITHMS

String matching (either exact or approximate) algorithms can be basically classified into three groups: single pattern matching algorithms (search for a single pattern within the text), algorithms that uses finite number of patterns (search finite number of patterns within the text) and algorithms that uses infinite number of patterns (search infinite number of patterns such as regular expressions within the text). Single pattern matching algorithms include Rabin-Karp [21], finite state automation based search [22], Knuth-Morris-Pratt [23] and Boyer-Moore [5]. Rabin-Karp algorithm can be used for both single pattern matching and for matching a finite number of patterns. For Rabin-Karp, in a string of length n, if p patterns of combined length m are to be matched, the best and worst case running time are O(n+m) and O(nm) respectively. Finite state automata based search is expensive to construct (power-set construction), but very easy to use. Knuth-Morris-Pratt (KMP) algorithm turns the search string into a finite state machine, and then runs the machine with the string to be searched as the input string. Running time of KMP algorithm is O(n + m). Algorithms used for finite number of patterns (multi patterns) include Aho-Corasick, Commentz-Water, Rabin-Karp and Wu-Manber and bit parallel algorithms [2]. They are common multiple pattern matching algorithms and in this paper we mainly focus on analyzing the time complexity among two algorithms: Aho-Corasick [3] and Commentz-Walter [4].

Aho-Corasick algorithm is a classic and scalable solution for exact string matching and is a widely used multi pattern matching algorithm. It has the running time of O(n+m+z) where n represents the length of the text, m represents the length of patterns and z represents the number of pattern occurrences in the text T (when the search tree is built off-line and compiled, it has the worst-run-time complexity of O(n + z) in space O(m)). Therefore, the time complexity of this algorithm is linear to the length of the patterns plus the length of the searched text plus the number of matches in the text. This is the major advantage of this algorithm. Aho-Corasick algorithm consists of two main phases: Finite State Machine (FSM) construction phase (Figure 1) and matching phase. In the FSM construction phase this algorithm uses the pattern set and first constructs the state machine and then considers failure links to avoid backtracking to root node when there is a failure. In the second phase it locates the pattern set within the given text if they occur in the string.

Boyer-Moore algorithm [5] is an efficient, single and exact pattern matching algorithm that preprocesses the pattern being searched for, but not the text being searched in. It searches for occurrences of pattern P in text T by applying shift rules: bad character rule and good suffix rule. Bad character rule deals with the amount of shifting along the pattern P to the left side when there is a mismatch character with the text T and good suffix rule deals with the amount of shift when there is a mismatch with a reoccurring suffix of the pattern. Shifting rules make this algorithm more efficient than other single pattern matching algorithms.

Commentz-Walter multi pattern matching algorithm combines the shifting methods of Boyer Moore with Aho-Corasick and therefore theoretically faster in locating a string than Aho-Corasick. Like Aho-Corasick, Commentz-Walter also consists of two phases: FSM construction and shift calculation phase (preprocessing) and matching phase. Here FSM is constructed reverse (reversed trie) (Figure 2) in order to use the shifting methods of Boyer-Moore algorithm. We have used 3000 sets of peptides to construct the state machine and then matched them against the known/unknown set of proteins and calculated and compared the time consumed by both Aho-Corasick and Commentz-Walter algorithms.

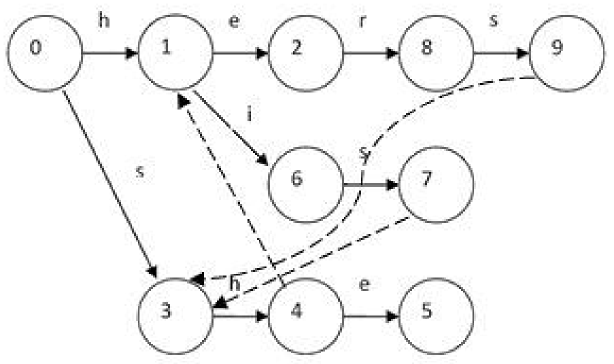

Figure 1: Aho-Corasick trie implementation for patterns he, she, his and hers

For example if we have four patterns he, hers, his and she, in both algorithms we start FSM creation with state 0(root node). If we add pattern ‘he’ first, in Aho-Corasick we create a new node (state 1), which represents character ‘h’, then we create another node (state 2), which represents character ‘e’ (Figure 1). Then we add pattern ‘she’. Since there is no node representing character ‘s’, we add a new node (state 3), then add two more nodes representing character ‘h’ and ‘e’. Likewise we add other two patterns to the state machine and finally we create failure links. In Commentz-Walter we add first pattern ‘he’ after reversing the keyword (Figure 2). We use the same method in the Commentz-Walter but finally we don’t create failure links (instead we use shifting whenever a failure occurs).

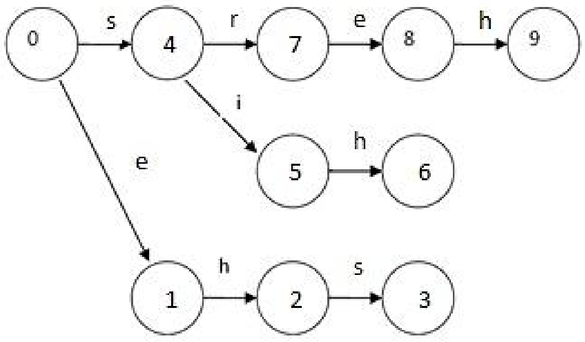

Figure 2: Commentz-Walter trie implementation for patterns he, she, his and hers

III. OBJECTIVES

As mentioned earlier Aho-Corasick is one of the commonly used exact multiple string matching algorithm and Commentz-Walter was introduced as a better alternative with a combination of ideas from both Aho-Corasick and Boyer-Moore. The main objective of this paper is to compare the time complexity of the two algorithms with a large set of peptide data for peptide identification.

IV. RELATED WORK

A comparative analysis has been done by Zeeshan et al. [8] where they have compared time complexity, search type and approach of five string multi-pattern matching algorithms: Aho-Corasick, Rabin-Karp, Bit Parallel, Commentz-Walter and Wu-Manber. However, they have not used real data for their comparison. They have mentioned that because of the difficulty in understanding the Commentz-Walter algorithm, it has not emerged as a popular string matching algorithm.

Experimental results of the Aho-Corasick, Wu-Manber, SBOM and Salmela's algorithms [19] were presented by Charalampos et al. where they run these algorithms against SWISS PROT amino acid database, amino acid and nucleotides sequences of the A-thaliana genome for the sets of size between 100 – 100000 patterns with the length of 8 and 32. Result showed that the algorithms are performing differently for different databases. According to that experiment, Commentz-Walter had the best performance on the FASTA nucleic acid database for more than 10000 patterns. Wu-Manber has performed the best for the FASTA Amino acid database for more than 10000 to 50000 patterns [14].

V. EXPERIMENTAL SETUP AND METHODS

We modified Komodia Aho-Corasick [6] for our purpose and developed our own implementation of Commentz-Walter algorithm in C++ programming language as we could not find any implementation in the literature. The pseudo-codes presented in Algorithm 1 and 2 are used in the implementations of the two algorithms concerned.

Algorithm 1: Aho-Corasick Pseudocode

```
1. for each pattern P=1 to N
2.    addToTheTrie();
3. end for
4. for each Protein R= 1 to M
5.    searchAhoCorasickMultiple();
6. end for
```

Algorithm 2: Commentz-Walter Pseudocode

```
1. createDepthShiftVector();
2. createSubStringShiftVector();
3. for each pattern P=1 to N
4.    reversePattern();
5.    addToTheTrie();
6. end for
7. for each Protein R= 1 to M
8.    searchCommentzWalterMultiple();
9. end for
```

Both algorithms were implemented in Visual Studio 2008 environment and run in Intel Core i7 2.2 GHz machine with 4GB RAM and 32bit OS. In this implementation, Aho Corasick algorithm uses predefined, different sets (sizes from 500 to 3000) of peptides to create the FSM (trie) first. For this FSM we have used a trie data structure which has several nodes representing states and each node consists of a failure node, a character, status of the node (whether a match can be found/not) and the depth from the root as parameters. Then a set of 100 Drosophila proteins extracted from UniProt database [7] was used as the input and this programme performed the multiple patterns matching and found all the patterns occurring within a protein in a single pass. When we have an input protein to be processed, first our matching is started from the state 0 (root node) and it is matched against the first amino acid in the protein. If it is matched we proceed to the next state and else we backtrack to the 0 state since 0 is the state of the failure node. Wherever we find any matching state, algorithm indicates the match and proceeds if it is not a leaf node. If it is a leaf node it indicates the match and stops.

In the Commentz-Walter implementation, we first created a reversed trie with predefined, different sizes of sets (from 500 to 3000) of peptides to create the reversed trie (FSM) where each node consists of state of the node, depth from the root, status of the node (whether a match can be found/not), and a character as parameters. Then we calculated the amount of shifts for all suffixes of patterns when there is a mismatch of a suffix (shift vector) and calculated the amount of shift when there is a mismatched character (depth vector). As mentioned earlier the same set of 100 proteins were used as the input and our program performed multiple pattern matching. In this algorithm first we find the minimum size of a peptide (Min) and then we align Min+1th position of the protein sequence with the root node and then start the matching process from the root. If a child of the root node gets matched with the amino acid aligned (Min th position) then we could proceed until mismatch or match is found (if a match is found we can go to the root and start, if mismatch is found we have to find the amount of suffix shift required), else we shift by the amount that is included in the shift vector.

In our experiments we used a maximum of 58 amino acids (a.a), a minimum of 4 a.a and an average of 14 a.a in length in the set of 500 peptides, a maximum of 82 amino acids a.a, a minimum of 4 a.a and an average of 14 a.a in length in the set of 1000 peptides, a maximum of 103 amino acids a.a, a minimum of 3 a.a and an average of 14 a.a in length in the set of 1500 peptides, a maximum of 125 amino acids a.a, a minimum of 3 a.a and an average of 14 a.a in length in the set of 2000 peptides, a maximum of 60 amino acids a.a, a minimum of 4 a.a and an average of 12 a.a in length in the set of 2500 peptides and used a maximum of 159 a.a, a minimum of 4 a.a and an average of 14 a.a in length in the set of 3000 peptides. We test with the 100 input proteins which have an average length of 531 a.a. We performed the experiments 10 times each in order to take the average.

Table 1: Time in milliseconds: Aho-Corasick Algorithm

Number of patterns in the FSM	Average Time consumed (ms)	
	Machine construction	Matching process
500	85	85
1000	186	186
1500	234	234
2000	327	327
2500	359	359
3000	542	542

Table 2: Time in milliseconds: Commentz-Walter Algorithm

Number of patterns in the FSM	Average Time consumed (ms)			
	Machine construction	Shift Vector	Depth vector	Matching process
500	795	112326	462	831
1000	1156	526018	147	2059
1500	1954	1230238	238	4265
2000	2579	2361684	331	5434
2500	2474	3325337	325	14056
3000	3722	6842905	543	15466

VI. RESULTS

Results show that the average times of 85, 186, 234, 327, 359 and 542 milliseconds were needed for 500, 1000, 1500, 2000, 2500 and 3000 peptides respectively (Table 1) in machine construction/ preprocessing phase of Aho-Corasick algorithm. In Commentz-Walter algorithm, preprocessing phase consists of three steps: constructing shift vector, constructing depths vector in case of mismatching and constructing the finite state machine. To construct the state machine, this algorithm requires average times of 795, 1156, 1954, 2579, 2474 and 3722 milliseconds (Table 2). These times required for state machine construction is higher than the time required for Aho-Corasick algorithm with the similar set of peptides. However the trie constructed in Commentz-Walter algorithm is different than the trie developed in Aho-Corasick algorithm because it is using reversed keywords. Shift vector construction is the critical step in Commentz-Walter algorithm since it consumes considerably higher amount of time (Table 2 column 3). It includes all the suffixes of patterns and amount of shifts when there is a mismatch of a suffix. Depth vector consists of the amount of shift when there is a mismatched character (Table 2 column 4). However, if we consider the matching time (Table 1 column 3 and Table 2 column 5) it also takes more time in Commentz-Walter algorithm than Aho-Corasick algorithm.

VII. CONCLUSIONS

According to the results presented, we could conclude that the Aho-Corasick algorithm is performing better in time consumed than Commentz-Walter for large sets of peptide data since Commentz-Walter algorithm needs more time for preprocessing (to calculate shifts, depths in case of mismatching and to construct the state machine).

VIII. FUTURE SCOPE

Parallelization of Aho-Corasick algorithm on an FPGA has already been performed [9][10][11][2]. Parallelizing string matching using multi-core architecture has been done by several research groups [12][13]. As future work, we plan to reduce the number of states in the trie (Aho-Corasick and Commentz-Walter) using optimization algorithms and parallelize these two algorithms in a multi-core architecture to improve their performance.

IX. REFERENCES

[1] Kal S., “Bioinformatics: Sequence alignment and Markov models”. McGraw-Hill Professional, 2008.

[2] Vidanagamachchi, S. M. Dewasurendra, S. D. Ragel R. G. Niranjan, M. "Tile optimization for area in FPGA based hardware acceleration of peptide identification". Industrial and Information Systems (ICIIS) 2011, 6th

IEEE International Conference, 16-19 Aug, pp: 140-145. doi: 10.1109/ICIINFS.2011.6038056

[3] A.V. AHO, M.J. CORASICK, “Efficient string matching: an aid to bibliographic search”. Communications of the ACM, Vol. 18, No. 6, 1979, pp: 333-340. doi: 10.1145/360825.360855

[4] Commentz-Walter, B. "A String Matching Algorithm Fast on the Average". Automata, Languages and Programming, Springer Berlin / Heidelberg, 1979, pp: 118-132

[5] Boyer Robert S., Moore J. STROTHER, "A fast string searching algorithm". Communications of the ACM, Vol. 20, No. 10, 1977, pp: 762-772. doi: 10.1145/359842.359859

[6] Komodia (2012), "Aho-Corasick source code". Available: http://www.komodia.com/aho-corasick

[7] Uniprot (2012), "Uniprot", Available: http://www.uniprot.org/

[8] Zeeshan A. KHAN, R.K. PATERIYA, " Multiple Pattern String Matching Methodologies: A Comparative Analysis". International Journal of Scientific and Research Publications (IJSRP), Vol. 2, No. 7, 2012, pp: 498-504

[9] Yoginder S. DANDASS, Shane C. BURGESS, Mark LAWRENCE and Susan M. BRIDGES, “Accelerating String Set Matching in FPGA Hardware for Bioinformatics Research”. BMC Bioinformatics 2008, Vol. 9, No. 197, 2008

[10] Jung, H. J. Baker, Z. K and Prasanna, V. K. “Performance of FPGA Implementation of Bit-split Architecture for Intrusion Detection Systems”. Proceedings of the 20th international conference on Parallel and distributed processing, ser. IPDPS’06, 2006, Washington, DC, USA: IEEE Computer Society, pp. 189–189.

[11] Liu, Y. Yang, Y. Liu, P. and Tan, J. “A table compression method for extended aho-corasick automaton”. Proceedings of the 14th International Conference on Implementation and Application of Automata, ser. CIAA ’09. 2009, Berlin, Heidelberg: Springer-Verlag, pp. 84–93. doi: 10.1007/978-3-642-02979-0_12

[12] Soewito, B. and Weng, N. "Methodology for Evaluating DNA Pattern Searching Algorithms on Multiprocessor". Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering, 2007, pp. 570 – 577. doi: 10.1109/BIBE.2007.4375618

[13] Tumeo, A. Villa, O. Chavarrı´a-Miranda, D.G. "Aho-Corasick String Matching on Shared and Distributed-Memory Parallel Architectures". IEEE Transactions on Parallel and Distributed Systems, 2012, pp. 436-443. doi: 10.1109/TPDS.2011.181

[14] Kouzinopoulos, C. S., Michailidis, P. D. and Margaritis, K. G. “Experimental Results on Multiple Pattern Matching Algorithms for Biological Sequences”. Bioinformatics'11, 2011, pp. 274-277.

[15] Dan Gusfield, “Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology”. Cambridge University Press, 1997.

[16] B. RAJU, S. DYLN, “Exact Multiple Pattern Matching Algorithm using DNA Sequence and Pattern Pair”. International Journal of Computer Applications, Vol. 17, No. 8, 2011, pp. 32-38

[17] Fisk, M. and Varghese, G. (2004) “Applying Fast String Matching to Intrusion Detection”. University of California San Diego, 2004

[18] S. BENFANO, M. ATU, W. NING and W. HAIBO, "High-speed string matching for network intrusion detection". International Journal of Communication Networks and Distributed Systems, Vol. 3, No. 4, 2009, pp. 319-339

[19] Leena Salmela, "Improved Algorithms for String Searching Problems", Helsinki University of Technology, 2009

[20] Textolution (2012), “FuzzySearch Library”. Available: http://www.textolution.com/fuzzysearch.asp

[21] K. RICHARD and R. MICHAEL, "Efficient randomized pattern-matching algorithms". IBM Journal of Research and Development, Vol. 31, No. 2, 1987, pp. 249-260

[22] String searching algorithm, "String searching algorithm". Available: http://en.wikipedia.org/wiki /String_searching_algorithm#Finite_state_automaton_based_search

[23] D. KNUTH, J. MORRIS and V. PRATT, "Fast pattern matching in strings". SIAM Journal on Computing, Vol. 6, No. 2, 1977, pp.323–350. doi: 10.1137/0206024

How to cite

S.M. Vidanagamachchi, S.D. Dewasurendra, R.G. Ragel, M.Niranjan, " Commentz-Walter: Any Better than Aho-Corasick for Peptide Identification?". *International Journal of Research in Computer Science*, 2 (6): pp. 33-37, November 2012. doi:10.7815/ijorcs.26.2012.053